

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

**A Serverless Distributed File System**

Inventor(s):

Atul Adya

William J. Bolosky

Gerald Cermak

John R. Douceur

Marvin Theimer

Roger Wattenhofer

1 **RELATED APPLICATIONS**

2 This application claims the benefit of U.S. Provisional Application No.  
3 60/278,905, filed March 26, 2001, entitled "A Serverless Distributed File System  
4 in an Untrusted Environment", to Atul Adya, Gerald Cermak, John R. Douceur,  
5 Marvin Theimer, Roger Wattenhofer, and William J. Bolosky, which is hereby  
6 incorporated by reference.

7  
8 **TECHNICAL FIELD**

9 This invention relates to computer networks and file systems, and  
10 particularly to a serverless distributed file system.

11  
12 **BACKGROUND**

13 File systems manage files and other data objects stored on computer  
14 systems. File systems were originally built into a computer's operating system to  
15 facilitate access to files stored locally on resident storage media. As computers  
16 became networked, some file storage capabilities were offloaded from individual  
17 user machines to special storage servers that stored large numbers of files on  
18 behalf of the user machines. When a file was needed, the user machine simply  
19 requested the file from the server. In this server-based architecture, the file system  
20 was extended to facilitate management of and access to files stored remotely at the  
21 storage server over a network.

22 Today, file storage is migrating toward a model in which files are stored on  
23 various networked computers, rather than on a central storage server. The  
24 serverless architecture poses new challenges to file systems. One particular  
25 challenge concerns managing files that are distributed over many different

1 computers in a manner that allows files to be reliably stored and accessible in spite  
2 of varying ones of the computers being inaccessible at any given time, while at the  
3 same time preventing access to the files by non-authorized users.

4 The invention addresses these challenges and provides solutions that are  
5 effective for serverless distributed file systems.

## 6 7 SUMMARY

8 A serverless distributed file system is described herein.

9 According to one aspect, files and directories are managed within the  
10 serverless distributed file system in different manners. Directories are managed  
11 using Byzantine-fault-tolerant groups, whereas files are managed without using  
12 Byzantine-fault-tolerant groups. This can result in improved performance as a  
13 result of storing fewer copies of a file than of the corresponding directory entry.

14 According to another aspect, the file system employs a hierarchical  
15 namespace to store files. The files are spread out across multiple computers, each  
16 of which can operate as both a client computer and a server computer, and each of  
17 which need not trust the others.

18 According to another aspect, responsibility for managing one or more  
19 directories in the file system are assigned to a directory group. Each member of  
20 the directory group is a computer participating in the system, and the directory  
21 group employs a plurality of locks to control access to objects (e.g., files and  
22 directories) in each directory. The locks include a first set of locks to control  
23 opening of the objects, and a second set of locks to control access to the data in the  
24 objects.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

The same numbers are used throughout the document to reference like components and/or features.

Fig. 1 illustrates an exemplary network environment that supports a serverless distributed file system.

Fig. 2 illustrates logical components of an exemplary computing device that is representative of any one of the devices of Fig. 1 that participate in the distributed file system.

Fig. 3 illustrates a more general computer environment which is used to implement the distributed file system of Fig. 1.

Fig. 4 illustrates an exemplary hierarchical namespace including a namespace root having multiple subtrees.

Fig. 5 is a flowchart illustrating an exemplary process for delegating management responsibility for a subtree to another directory group.

Fig. 6 is a flowchart illustrating an exemplary process for looking up the directory group responsible for managing a particular pathname.

Fig. 7 illustrates the exemplary storage of a file and corresponding directory entry in a serverless distributed file system.

Fig. 8 is a flowchart illustrating an exemplary process for storing a file in a serverless distributed file system.

Fig. 9 is a flowchart illustrating an exemplary process for determining whether to allow a particular object to be opened.

## DETAILED DESCRIPTION

The following discussion is directed to a symbiotic, serverless, distributed file system that runs on multiple networked computers and stores files across the computers rather than on a central server or cluster of servers. The symbiotic nature implies that the machines cooperate but do not completely trust one another. The file system does not manage the disk storage directly, but rather relies on existing file systems on local machines, such as those file systems integrated into operating systems (e.g., the Windows NT® file system).

The discussions herein make reference to public key cryptography, encryption, and digital signatures. Generally, public key cryptography refers to the use of public and private keys, collectively referred to as a key pair. An entity (e.g., a user, a software application, etc.) keeps the private key secret, but makes the public key available to others. Data, typically referred to as plaintext, can be encrypted using an encryption algorithm and the public key in such a way that the encrypted result (typically referred to as ciphertext) cannot be easily decrypted without knowledge of the corresponding private key, but can be relatively easily decrypted with knowledge of the corresponding private key. Similarly, data can be digitally signed using an encryption algorithm and the private key in such a way that the signature can be easily verified using the corresponding public key, but a signature cannot easily be generated without the private key. The discussions herein assume a basic understanding of cryptography by the reader. For a basic introduction of cryptography, the reader is directed to a text written by Bruce Schneier and entitled "Applied Cryptography: Protocols, Algorithms, and Source Code in C," published by John Wiley & Sons with copyright 1994 (or second edition with copyright 1996).

## Serverless Distributed File System

Fig. 1 illustrates an exemplary network environment 100 that supports a serverless distributed file system. Four client computing devices 102, 104, 106, and 108 are coupled together via a data communications network 110. Although four computing devices are illustrated, different numbers (either greater or fewer than four) may be included in network environment 100.

Network 110 represents any of a wide variety of data communications networks. Network 110 may include public portions (e.g., the Internet) as well as private portions (e.g., an internal corporate Local Area Network (LAN)), as well as combinations of public and private portions. Network 110 may be implemented using any one or more of a wide variety of conventional communications media including both wired and wireless media. Any of a wide variety of communications protocols can be used to communicate data via network 110, including both public and proprietary protocols. Examples of such protocols include TCP/IP, IPX/SPX, NetBEUI, etc.

Computing devices 102-108 represent any of a wide range of computing devices, and each device may be the same or different. By way of example, devices 102-108 may be desktop computers, laptop computers, handheld or pocket computers, personal digital assistants (PDAs), cellular phones, Internet appliances, consumer electronics devices, gaming consoles, and so forth.

Two or more of devices 102-108 operate to implement a serverless distributed file system. The actual devices participating in the serverless distributed file system can change over time, allowing new devices to be added to the system and other devices to be removed from the system. Each device 102-

1 106 that implements (participates in) the distributed file system has portions of its  
2 mass storage device(s) (e.g., hard disk drive) allocated for use as either local  
3 storage or distributed storage. The local storage is used for data that the user  
4 desires to store on his or her local machine and not in the distributed file system  
5 structure. The distributed storage portion is used for data that the user of the  
6 device (or another device) desires to store within the distributed file system  
7 structure.

8 In the illustrated example of Fig. 1, certain devices connected to network  
9 110 have one or more mass storage devices that include both a distributed portion  
10 and a local portion. The amount allocated to distributed or local storage varies  
11 among the devices. For example, device 102 has a larger percentage allocated for  
12 a distributed system portion 120 in comparison to the local portion 122; device  
13 104 includes a distributed system portion 124 that is approximately the same size  
14 as the local portion 126; and device 106 has a smaller percentage allocated for a  
15 distributed system portion 128 in comparison to the local portion 130. The storage  
16 separation into multiple portions may occur on a per storage device basis (e.g., one  
17 hard drive is designated for use in the distributed system while another is  
18 designated solely for local use), and/or within a single storage device (e.g., part of  
19 one hard drive may be designated for use in the distributed system while another  
20 part is designated for local use). The amount allocated to distributed or local  
21 storage may vary over time. Other devices connected to network 110, such as  
22 computing device 108, may not implement any of the distributed file system and  
23 thus do not have any of their mass storage device(s) allocated for use by the  
24 distributed system. Hence, device 108 has only a local portion 132.

1 A distributed file system 150 operates to store one or more copies of files  
2 on different computing devices 102-106. When a new file is created by the user of  
3 a computer, he or she has the option of storing the file on the local portion of his  
4 or her computing device, or alternatively in the distributed file system. If the file  
5 is stored in the distributed file system 150, the file will be stored in the distributed  
6 system portion of the mass storage device(s) of one or more of devices 102-106.  
7 The user creating the file typically has no ability to control which device 102-106  
8 the file is stored on, nor any knowledge of which device 102-106 the file is stored  
9 on. Additionally, replicated copies of the file will typically be saved, allowing the  
10 user to subsequently retrieve the file even if one of the computing devices 102-106  
11 on which the file is saved is unavailable (e.g., is powered-down, is malfunctioning,  
12 etc.).

13 The distributed file system 150 is implemented by one or more components  
14 on each of the devices 102-106, thereby obviating the need for any centralized  
15 server to coordinate the file system. These components operate to determine  
16 where particular files are stored, how many copies of the files are created for  
17 storage on different devices, and so forth. Exactly which device will store which  
18 files depends on numerous factors, including the number of devices in the  
19 distributed file system, the storage space allocated to the file system from each of  
20 the devices, how many copies of the file are to be saved, a cryptographically  
21 secure random number, the number of files already stored on the devices, and so  
22 on. Thus, the distributed file system allows the user to create and access files (as  
23 well as folders or directories) without any knowledge of exactly which other  
24 computing device(s) the file is being stored on.  
25



1 Distributed file system 150 is designed to be scalable to support large  
2 numbers of computers within system 150. Protocols and data structures used by  
3 the components on the devices in system 150 are designed so as not to be  
4 proportional to the number of computers in the system, thereby allowing them to  
5 readily scale to large numbers of computers.

6 The files stored by the file system are distributed among the various devices  
7 102-106 and stored in encrypted form. When a new file is created, the device on  
8 which the file is being created encrypts the file prior to communicating the file to  
9 other device(s) for storage. The directory entry (which includes the file name) for  
10 a new file is also communicated to other device(s) for storage, which need not be  
11 (and typically will not be) the same device(s) on which the encrypted file is stored.  
12 Additionally, if a new folder or directory is created, the directory entry (which  
13 includes the folder name or directory name) is also communicated to the other  
14 device(s) for storage. As used herein, a directory entry refers to any entry that can  
15 be added to a file system directory, including both file names and directory (or  
16 folder) names.

17 The distributed file system 150 is designed to prevent unauthorized users  
18 from reading data stored on one of the devices 102-106. Thus, a file created by  
19 device 102 and stored on device 104 is not readable by the user of device 104  
20 (unless he or she is authorized to do so). In order to implement such security, the  
21 contents of files as well as all file and directory names in directory entries are  
22 encrypted, and only authorized users are given the decryption key. Thus, although  
23 device 104 may store a file created by device 102, if the user of device 104 is not  
24 an authorized user of the file, the user of device 104 cannot decrypt (and thus  
25 cannot read) either the contents of the file or the file name in its directory entry.

1 The distributed file system 150 employs a hierarchical storage structure,  
2 having one or more namespace roots as well as multiple subtrees under each  
3 namespace root. The management of different subtrees can be delegated to  
4 different groups of computers, thereby preventing the computers managing a  
5 namespace root or a particular subtree(s) from becoming overburdened.

6 The distributed file system 150 also manages the storage of files and the  
7 directory entries corresponding to those files differently. A file being stored in  
8 system 150 is replicated and saved on multiple different computers in the system.  
9 Additionally, a directory entry is generated for the file and is also saved on  
10 multiple different computers in the system. A larger number of directory entry  
11 copies are saved than are file copies. In one implementation, the directory entries  
12 are stored on computers that are part of a Byzantine-fault-tolerant group, as  
13 discussed in more detail below.

14 The distributed file system 150 also employs a directory and file lock  
15 mechanism that allows control over who may read or write directories and files.  
16 When used with computers in a Byzantine group, the lock mechanism employed  
17 attempts to increase performance by increasing the number of operations that can  
18 be performed locally without requiring action by the directory group, as discussed  
19 in more detail below.

20 Every computer 102 – 106 in distributed file system 150 can have three  
21 functions: it can be a client for a local user, it can be a repository for encrypted  
22 copies of files stored in the system, and it can be a member of a group of  
23 computers that maintain one or more directories.

24 Generally, when a user on a computer 102 – 106 opens a file in a given  
25 directory, the computer sends a request to a set of computers that collectively

1 manage that directory (called a "Byzantine group" or "directory group") using a  
2 Byzantine-fault-tolerant protocol. The Byzantine group grants a file lock to the  
3 computer, allowing it to make local updates to the file (if it is a write lock) and to  
4 subsequently push those updates back to the Byzantine group. If the computer has  
5 accessed this file recently, it will probably have an encrypted copy of the file  
6 contents in a local cache, so it need only retrieve the cached copy and decrypt it,  
7 after which it can begin reading or writing the file. If it has not accessed the  
8 current version of the file recently, the computer retrieves an encrypted copy of the  
9 file from one of the computers that stores the file. The information about which  
10 computers hold current copies is provided by the Byzantine group along with the  
11 lock grant; if one or more of the file-storage computers are down, the computer  
12 retrieves the file from a different one. The Byzantine group also provides a  
13 cryptographic hash of the file contents that the computer uses to validate the file it  
14 fetches.

### 15 16 **File Encryption**

17 The files are encrypted using a technology known as "convergent  
18 encryption". Convergent encryption has the following two properties. First, if  
19 two or more encryptable objects are identical, then even if different encryption  
20 keys are used to encrypt them to provide individual cipher objects, one does not  
21 need to have access to any of the encryption keys to determine from an  
22 examination of the cipher objects that the encryptable objects are identical.  
23 Second, if two or more encryptable objects are identical but are encrypted with  
24 different encryption keys, the total space that is required to store all of the cipher  
25

1 objects is proportional to the space that is required to store a single encryptable  
2 object, plus a constant amount of storage for each distinct encryption key.

3 Generally, according to convergent encryption, a file  $F$  (or any other type of  
4 encryptable object) is initially hashed using a one-way hashing function  $h$  (e.g.,  
5 SHA, MD5, etc.) to produce a hash value  $h(F)$ . The file  $F$  is then encrypted using  
6 a symmetric cipher (e.g., RC4, RC2, etc.) with the hash value as the key, or  
7  $E_{h(F)}(F)$ . Next, read access control entries are created for each authorized user who  
8 is granted read access to the encrypted file. Write access control is governed by  
9 the directory server that stores the directory entry for the file. The read access  
10 control entries are formed by encrypting the file's hash value  $h(F)$  with any  
11 number of keys  $K_1, K_2, \dots, K_m$ , to yield  $E_{K_1}(h(F)), E_{K_2}(h(F)), \dots, E_{K_m}(h(F))$ . In  
12 one implementation, each key  $K$  is the user's public key of a public/private key  
13 pair for an asymmetric cipher (e.g., RSA).

14 With convergent encryption, one encrypted version of the file is stored and  
15 replicated among the serverless distributed file system 150. Along with the  
16 encrypted version of the file is stored one or more access control entries depending  
17 upon the number of authorized users who have access. Thus, a file in the  
18 distributed file system 150 has the following structure:

$$19 \\ 20 [E_{h(F)}(F), \langle E_{K_1}(h(F)) \rangle, \langle E_{K_2}(h(F)) \rangle, \dots, \langle E_{K_m}(h(F)) \rangle]$$

21  
22 One advantage of convergent encryption is that the encrypted file can be  
23 evaluated by the file system to determine whether it is identical to another file  
24 without resorting to any decryption (and hence, without knowledge of any  
25 encryption keys). Unwanted duplicative files can be removed by adding the

1 authorized user(s) access control entries to the remaining file. Another advantage  
2 is that the access control entries are very small in size, on the order of bytes as  
3 compared to possibly gigabytes for the encrypted file. As a result, the amount of  
4 overhead information that is stored in each file is small. This enables the property  
5 that the total space used to store the file is proportional to the space that is required  
6 to store a single encrypted file, plus a constant amount of storage for each  
7 additional authorized reader of the file.

8 For more information on convergent encryption, the reader is directed to  
9 co-pending U.S. Patent Application Serial No. 09/565,821, entitled "Encryption  
10 Systems and Methods for Identifying and Coalescing Identical Objects Encrypted  
11 with Different Keys", which was filed May 5, 2000, in the names of Douceur et  
12 al., and is commonly assigned to Microsoft Corporation. This application is  
13 hereby incorporated by reference.

### 14 Directory Entry Encryption

15 The file and directory names within directory entries are encrypted using a  
16 process referred to as "exclusive encryption". Exclusive encryption allows the file  
17 and directory names within directory entries to be stored in an encrypted form,  
18 thereby preventing unauthorized users from improperly gaining any information  
19 based on the name of a file or directory. Additionally, exclusive encryption has  
20 the following three properties. First, no two encrypted entries in a directory will  
21 decrypt to the same name. Second, all encrypted entries in a directory decrypt to  
22 syntactically legal names. Third, the directory group that maintains the directory  
23 does not have access to the plaintext names of the entries. Thus, file system 150 is  
24 able to ensure both that no two entries in a directory are encryptions of the same  
25

1 name and that all entries in a directory are encryptions of syntactically legal  
2 names, while at the same time ensuring that the device maintaining the directory  
3 does not have access to the plaintext names of the entries.

4 Generally, according to exclusive encryption, a plaintext name (the file or  
5 directory name within the directory entry) is mapped to a new name. The mapped  
6 name is optionally decasified into a decasified (case-insensitive) name and  
7 corresponding case information, allowing duplicate name detection to be case-  
8 insensitive. The mapped (and optionally decasified) name is then encoded and  
9 encrypted. This encrypted name (and optionally accompanying case information)  
10 are forwarded to the directory group that is responsible for managing the directory  
11 entry (e.g., based on pathname, as discussed in more detail below).

12 For more information on exclusive encryption, the reader is directed to co-  
13 pending U.S. Patent Application Serial No. 09/764,962, entitled "Exclusive  
14 Encryption for a Secure Directory Service", which was filed January 17, 2001, in  
15 the names of Douceur et al., and is commonly assigned to Microsoft Corporation.  
16 This application is hereby incorporated by reference.

### 17 18 **File Format**

19 The file format for serverless distributed file system 150 of Fig. 1 is  
20 composed of two parts: a primary data stream and a metadata stream. The  
21 primary data stream contains a file that is divided into multiple blocks. Each  
22 block is encrypted using a symmetric cipher (e.g., RC4) and a hash of the block as  
23 the encryption key. The metadata stream contains a header, a structure for  
24 indexing the encrypted blocks in the primary data stream, and some user  
25 information.

1       The indexing tree structure defines leaf nodes for each of the blocks. Each  
2 leaf node consists of an access value used for decryption of the associated block  
3 and a verification value used to verify the encrypted block independently of other  
4 blocks. In one implementation, the access value is formed by hashing the file  
5 block and encrypting the resultant hash value using a symmetric cipher and a  
6 randomly generated key. The key is then encrypted using an asymmetric cipher  
7 (e.g., RSA) and the user's public key as the encryption key. The verification value  
8 is formed by hashing the associated encrypted block using a one-way hash  
9 function (e.g., SHA).

10       Depending on the size of the file, the indexing structure may include  
11 intermediate nodes formed by grouping the leaf nodes into tree blocks and  
12 computing hash values of each tree block. These intermediate nodes can again be  
13 segmented into blocks and each block hashed to form the next nodes. This can be  
14 repeated as many times as desired until reaching a root node. The root node is  
15 then hashed, and the hash value is used along with the metadata header and user  
16 information to produce a verification value for the entire file. In one  
17 implementation, the whole-file verification value is signed with a user's signature.  
18 Alternatively, a file may be constructed without such signatures.

19       The file format supports verification of individual file blocks without  
20 knowledge of the randomly generated key or any user keys. To verify a block of  
21 the file, the file system optionally evaluates the signature on whole file verification  
22 value (if one exists), checks that the whole-file verification value matches the hash  
23 of the root block, metadata header and user information and then traverses the tree  
24 to the appropriate leaf node associated with a target block to be verified. The file  
25

1 system hashes the target block and if the hash matches the access value contained  
2 in the leaf node, the block is authentic.

3 The file format further supports reading from and writing to individual  
4 blocks without interfering with other blocks. The file format is also conducive for  
5 sparse files that have vast areas of non-data.

6 For more information on the file format, the reader is directed to co-  
7 pending U.S. Patent Application Serial No. 09/814,259, entitled "On-Disk File  
8 Format for a Serverless Distributed File System", which was filed March 21, 2001,  
9 in the names of Bolosky et al., and is commonly assigned to Microsoft  
10 Corporation. This application is hereby incorporated by reference.

### 11 12 Computing Device Architecture

13 Fig. 2 illustrates logical components of an exemplary computing device 200  
14 that is representative of any one of the devices 102-106 of Fig. 1 that participate in  
15 the distributed file system 150. Computing device 200 includes a server  
16 component 202, a client component 204, a memory 206, a mass storage device  
17 208, and a distributed file system interface 210. Computing device 200 also  
18 typically includes additional components (e.g., a processor), however these  
19 additional components have not been shown in Fig. 2 so as not to clutter the  
20 drawings. A more general description of a computer architecture with various  
21 hardware and software components is described below with reference to Fig. 3.

22 Memory 206 can be any of a wide variety of conventional volatile and/or  
23 nonvolatile memories, such as RAM, ROM, Flash memory, and so on. Mass  
24 storage device 208 can be any of a wide variety of conventional nonvolatile  
25 storage devices, such as a magnetic disk, optical disk, Flash memory, and so forth.



1 Mass storage device 208 is partitioned into a distributed system portion and a local  
2 portion. Although only one mass storage device 208 is illustrated in Fig. 2,  
3 computing device 200 may include multiple storage devices 208 (of different  
4 types, or alternatively all of the same type).

5 Computing device 200 is intended to be used in a serverless distributed file  
6 system, and as such includes both a server component 202 and client component  
7 204. Server component 202 handles requests when device 200 is responding to a  
8 request involving a file or directory entry stored (or to be stored) in storage device  
9 208, while client component 204 handles the issuance of requests by device 200  
10 for files or directories stored (or to be stored) in the distributed file system. Client  
11 component 204 and server component 202 operate independently of one another.  
12 Thus, situations can arise where the serverless distributed file system 150 causes  
13 files being stored by client component 204 to be stored in mass storage device 208  
14 by server component 202.

15 Client component 204 includes a storage and retrieval control module 220,  
16 which along with interface 210, manages access to the serverless distributed file  
17 system 150 for the creation, storage, retrieval, reading, writing, modifying, and  
18 verifying of files and directories on behalf of computing device 150. Control  
19 module 220 uses a directory group lookup module 222 to identify a directory  
20 group that is responsible for managing a particular file or directory, a file  
21 encryption module 226 to encrypt files, and a directory encryption module 228 to  
22 encrypt file and directory names in directory entries. The operation of these  
23 modules is discussed in more detail below.

24 The server component 202 includes a distributed system control module  
25 250, a duplication identifier 252, and a subtree delegation module 254.

1 Distributed system control module 250 manages access to the encrypted files 240.  
2 It communicates with mass storage device 208 to store and retrieve encrypted files  
3 240. Distributed system control module 250 also maintains a record of the  
4 directory entries (not shown) in memory 206 and/or mass storage device 208 that  
5 are stored at computing device 200 (or alternatively that are stored elsewhere in  
6 the serverless distributed file system). Subtree delegation module 254 operates to  
7 delegate subtrees to other directory groups, as discussed in more detail below.

8 Duplication identifier 252 helps identify identical encrypted files in the  
9 distributed file system. When the duplication identifier 252 finds a duplication  
10 that is not an intentional replication for fault tolerant purposes, the duplication  
11 identifier 252 notifies the control module 250, which then eliminates the  
12 duplicated file and adds the access control entries to the eliminated file to the  
13 remaining file.

14 Fig. 3 illustrates a more general computer environment 300, which is used  
15 to implement the distributed file system. The computer environment 300 is only  
16 one example of a computing environment and is not intended to suggest any  
17 limitation as to the scope of use or functionality of the computer and network  
18 architectures. Neither should the computer environment 300 be interpreted as  
19 having any requirement regarding the inclusion (or exclusion) of any components  
20 or the coupling or combination of components illustrated in the exemplary  
21 computer environment 300.

22 Computer environment 300 includes a general-purpose computing device in  
23 the form of a computer 302. The components of computer 302 can include, by are  
24 not limited to, one or more processors or processing units 304, a system memory  
25

1 306, and a system bus 308 that couples various system components including the  
2 processor 304 to the system memory 306.

3 The system bus 308 represents one or more of any of several types of bus  
4 structures, including a memory bus or memory controller, a peripheral bus, an  
5 accelerated graphics port, and a processor or local bus using any of a variety of  
6 bus architectures. By way of example, such architectures can include an Industry  
7 Standard Architecture (ISA) bus, a Micro Channel Architecture (MCA) bus, an  
8 Enhanced ISA (EISA) bus, a Video Electronics Standards Association (VESA)  
9 local bus, and a Peripheral Component Interconnects (PCI) bus also known as a  
10 Mezzanine bus.

11 Computer 302 typically includes a variety of computer readable media.  
12 Such media can be any available media that is accessible by computer 302 and  
13 includes both volatile and non-volatile media, removable and non-removable  
14 media.

15 The system memory 306 includes computer readable media in the form of  
16 volatile memory, such as random access memory (RAM) 310, and/or non-volatile  
17 memory, such as read only memory (ROM) 312. A basic input/output system  
18 (BIOS) 314, containing the basic routines that help to transfer information  
19 between elements within computer 302, such as during start-up, is stored in ROM  
20 312. RAM 310 typically contains data and/or program modules that are  
21 immediately accessible to and/or presently operated on by the processing unit 304.

22 Computer 302 may also include other removable/non-removable,  
23 volatile/non-volatile computer storage media. By way of example, Fig. 3  
24 illustrates a hard disk drive 316 for reading from and writing to a non-removable,  
25 non-volatile magnetic media (not shown), a magnetic disk drive 318 for reading

1 from and writing to a removable, non-volatile magnetic disk 320 (e.g., a "floppy  
2 disk"), and an optical disk drive 322 for reading from and/or writing to a  
3 removable, non-volatile optical disk 324 such as a CD-ROM, DVD-ROM, or other  
4 optical media. The hard disk drive 316, magnetic disk drive 318, and optical disk  
5 drive 322 are each connected to the system bus 308 by one or more data media  
6 interfaces 326. Alternatively, the hard disk drive 316, magnetic disk drive 318,  
7 and optical disk drive 322 can be connected to the system bus 308 by one or more  
8 interfaces (not shown).

9 The disk drives and their associated computer-readable media provide non-  
10 volatile storage of computer readable instructions, data structures, program  
11 modules, and other data for computer 302. Although the example illustrates a  
12 hard disk 316, a removable magnetic disk 320, and a removable optical disk 324,  
13 it is to be appreciated that other types of computer readable media which can store  
14 data that is accessible by a computer, such as magnetic cassettes or other magnetic  
15 storage devices, flash memory cards, CD-ROM, digital versatile disks (DVD) or  
16 other optical storage, random access memories (RAM), read only memories  
17 (ROM), electrically erasable programmable read-only memory (EEPROM), and  
18 the like, can also be utilized to implement the exemplary computing system and  
19 environment.

20 Any number of program modules can be stored on the hard disk 316,  
21 magnetic disk 320, optical disk 324, ROM 312, and/or RAM 310, including by  
22 way of example, an operating system 326, one or more application programs 328,  
23 other program modules 330, and program data 332. Each of such operating  
24 system 326, one or more application programs 328, other program modules 330,  
25

1 and program data 332 (or some combination thereof) may implement all or part of  
2 the resident components that support the distributed file system.

3 A user can enter commands and information into computer 302 via input  
4 devices such as a keyboard 334 and a pointing device 336 (e.g., a "mouse").  
5 Other input devices 338 (not shown specifically) may include a microphone,  
6 joystick, game pad, satellite dish, serial port, scanner, and/or the like. These and  
7 other input devices are connected to the processing unit 304 via input/output  
8 interfaces 340 that are coupled to the system bus 308, but may be connected by  
9 other interface and bus structures, such as a parallel port, game port, or a universal  
10 serial bus (USB).

11 A monitor 342 or other type of display device can also be connected to the  
12 system bus 308 via an interface, such as a video adapter 344. In addition to the  
13 monitor 342, other output peripheral devices can include components such as  
14 speakers (not shown) and a printer 346 which can be connected to computer 302  
15 via the input/output interfaces 340.

16 Computer 302 can operate in a networked environment using logical  
17 connections to one or more remote computers, such as a remote computing device  
18 348. By way of example, the remote computing device 348 can be a personal  
19 computer, portable computer, a server, a router, a network computer, a peer device  
20 or other common network node, and the like. The remote computing device 348 is  
21 illustrated as a portable computer that can include many or all of the elements and  
22 features described herein relative to computer 302.

23 Logical connections between computer 302 and the remote computer 348  
24 are depicted as a local area network (LAN) 350 and a general wide area network  
25

1 (WAN) 352. Such networking environments are commonplace in offices,  
2 enterprise-wide computer networks, intranets, and the Internet.

3 When implemented in a LAN networking environment, the computer 302 is  
4 connected to a local network 350 via a network interface or adapter 354. When  
5 implemented in a WAN networking environment, the computer 302 typically  
6 includes a modem 356 or other means for establishing communications over the  
7 wide network 352. The modem 356, which can be internal or external to computer  
8 302, can be connected to the system bus 308 via the input/output interfaces 340 or  
9 other appropriate mechanisms. It is to be appreciated that the illustrated network  
10 connections are exemplary and that other means of establishing communication  
11 link(s) between the computers 302 and 348 can be employed.

12 In a networked environment, such as that illustrated with computing  
13 environment 300, program modules depicted relative to the computer 302, or  
14 portions thereof, may be stored in a remote memory storage device. By way of  
15 example, remote application programs 358 reside on a memory device of remote  
16 computer 348. For purposes of illustration, application programs and other  
17 executable program components such as the operating system are illustrated herein  
18 as discrete blocks, although it is recognized that such programs and components  
19 reside at various times in different storage components of the computing device  
20 302, and are executed by the data processor(s) of the computer.

21 An implementation of the distributed file system 150 may be described in  
22 the general context of computer-executable instructions, such as program modules,  
23 executed by one or more computers or other devices. Generally, program modules  
24 include routines, programs, objects, components, data structures, etc. that perform  
25 particular tasks or implement particular abstract data types. Typically, the

1 functionality of the program modules may be combined or distributed as desired in  
2 various embodiments.

3 An implementation of the file format for the encrypted files may be stored  
4 on or transmitted across some form of computer readable media. Computer  
5 readable media can be any available media that can be accessed by a computer.  
6 By way of example, and not limitation, computer readable media may comprise  
7 "computer storage media" and "communications media."

8 "Computer storage media" include volatile and non-volatile, removable and  
9 non-removable media implemented in any method or technology for storage of  
10 information such as computer readable instructions, data structures, program  
11 modules, or other data. Computer storage media include, but are not limited to,  
12 RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM,  
13 digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic  
14 tape, magnetic disk storage or other magnetic storage devices, or any other  
15 medium which can be used to store the desired information and which can be  
16 accessed by a computer.

17 "Communication media" typically embody computer readable instructions,  
18 data structures, program modules, or other data in a modulated data signal, such as  
19 carrier wave or other transport mechanism. Communication media also include  
20 any information delivery media. The term "modulated data signal" means a signal  
21 that has one or more of its characteristics set or changed in such a manner as to  
22 encode information in the signal. By way of example, and not limitation,  
23 communication media include wired media such as a wired network or direct-  
24 wired connection, and wireless media such as acoustic, RF, infrared, and other  
25

1 wireless media. Combinations of any of the above are also included within the  
2 scope of computer readable media.

### 3 4 **Hierarchical Storage Structure**

5 Distributed file system 150 employs a hierarchical file storage structure  
6 including one or more namespace roots each capable of supporting one or more  
7 subtrees of directories or folders, and with each subtree being capable of  
8 supporting one or more additional subtrees. A directory can be viewed as a  
9 simulated file folder, being capable of holding zero or more files and/or zero or  
10 more other directories. A subtree refers to one or more directories and includes a  
11 root (it may also include a namespace root), and has the property that the path  
12 from the subtree root to all members of the subtree is within the subtree itself.  
13 Fig. 4 illustrates an exemplary hierarchical namespace 400 including a namespace  
14 root having multiple subtrees including directories A, B, C, D, E, F, G, H, J, I, M,  
15 K, and L. Although many more directories will typically be included in subtrees  
16 of a namespace root, only a few have been illustrated in Fig. 4 for ease of  
17 explanation.

18 Each subtree is managed by a group of one or more computers referred to  
19 as a directory group. Although discussed herein primarily as directory groups  
20 managing subtrees, alternatively one or more directory groups may manage an  
21 arbitrary set of directories within the namespace. One or more modules of the  
22 computer are responsible for implementing directory services to manage the  
23 subtree(s) it is assigned, such as control module 250 of Fig. 2. In one  
24 implementation, each directory group is a Byzantine-fault-tolerant group (or  
25 simply referred to as a Byzantine group), as discussed in more detail below.



1 However, directory groups need not be Byzantine-fault-tolerant groups, and other  
2 groupings can be used.

3 The solid lines in Fig. 4 illustrate relationships between directories,  
4 identifying which directories are sub-directories of which other directories. For  
5 example, directory C is a sub-directory of directory B. A directory can also be  
6 referred to as the "parent" directory of any of its sub-directories. For example,  
7 directory B can be referred to as the parent directory of directory C.

8 Each dashed box in Fig. 4 illustrates a directory group that manages the  
9 directories included within the particular dashed line. Thus, in the example  
10 namespace 400, the root namespace is managed by a directory group 402,  
11 directories A, B, C, F, and G are managed by a directory group 404, directories D  
12 and E are managed by a directory group 406, directories H and J are managed by a  
13 directory group 408, and directories K, I, L, and M are managed by a directory  
14 group 410.

15 A directory group managing a particular directory or namespace is  
16 responsible for maintaining a directory entry for each file stored in that directory,  
17 as well as a directory entry for each sub-directory within the directory. Each  
18 directory entry for a file identifies one or more computers in the distributed file  
19 system 150 where the file is stored. Each directory entry for a sub-directory  
20 identifies the directory group responsible for managing that sub-directory.  
21 Directory entries may also contain additional information, such as: creation,  
22 modification and access time stamps; read and write access control lists; the set of  
23 replica locations; the size of the file; and so forth.

24 Each directory group is responsible for managing a namespace root and/or  
25 one or more subtrees within the namespace. Each directory group is further able

1 to identify one or more additional subtrees and delegate management  
2 responsibility for those additional subtrees to another directory group. For  
3 example, directories D and E may have originally been managed by directory  
4 group 404, but subsequently delegated to directory group 406.

5 A directory group can decide at any time to delegate a subtree to another  
6 directory group. In one implementation, this decision is based on workload, and  
7 the directory group decides to delegate a subtree when the group determines that it  
8 is becoming overloaded. Various factors can be used by a group to determine  
9 when it is becoming overloaded, and in one exemplary implementation each  
10 directory group tries to manage a subtree of size approximately equal to the mean  
11 count of expected directories per machine (e.g., on the order of 10,000).

12 The directory group to which the subtree is to be delegated can be  
13 determined in a variety of manners. In one implementation, the directory group  
14 performing the delegation selects randomly from the computers in distributed file  
15 system 150 that it is aware of, and uses those selected computers as the new  
16 directory group to which the subtree is to be delegated. Various other factors may  
17 weigh into the selection process (e.g., not selecting those computers that have low  
18 availability, not selecting those computers that have recently delegated a subtree,  
19 etc.).

20 A directory group is able to delegate a particular subtree by generating a  
21 delegation certificate that is digitally signed by one or more members of the  
22 directory group. In situations where multiple members sign a delegation  
23 certificate, the signature process can take various forms. In one implementation,  
24 each member signs its own copy of the delegation certificate. In another  
25 implementation, the delegation certificate is recursively signed (e.g., the certificate

1 is signed by one member, and then the digitally signed certificate is signed by  
2 another member, etc.). The order in which different members recursively sign the  
3 certificate does not matter, so long as the order is known to the verifier when  
4 verifying the digital signature (e.g., the verifier may be pre-programmed with the  
5 order of signature, or information identifying the order may be included in the  
6 certificate). The following illustrates an exemplary certificate recursively signed  
7 by four signers:

$$\sigma_{S4}(\sigma_{S3}(\sigma_{S2}(\sigma_{S1}(DC))))$$

9 where DC represents the delegation certificate being digitally signed, and  $\sigma_{Si}()$   
10 indicates that the contents of  $()$  have been digitally signed by signer  $i$ .

11 In one implementation, the number of members (computers) in a directory  
12 group is dependent on the number of faulty computers that the designer desires to  
13 be able to tolerate. As used herein, a faulty computer refers to a computer that is  
14 either inaccessible (e.g., the computer has been powered off or is malfunctioning)  
15 or that has been corrupted (e.g., a malicious user or program has gained access to  
16 the computer and is able to respond to queries inappropriately, such as by not  
17 giving proper response or giving improper data). In one specific example, in order  
18 to tolerate  $f$  faulty computers, a directory group includes  $3f+1$  computers.  
19 Additionally, in this example, at least  $f+1$  computers digitally sign the delegation  
20 certificate.

21 Each namespace root has associated with it a certificate that is obtained  
22 from a certification authority (CA). The certification authority is a trusted  
23 authority that verifies the creation of the namespace. Each delegation certificate  
24 associated with a subtree includes a certificate chain that traces from the current  
25 subtree back up through zero or more other subtrees to the namespace root

1 certificate signed by the CA. Thus, each delegation certificate has associated with  
2 it multiple certificates that prove it is the authorized directory group for managing  
3 the subtree (by establishing a certificate chain back to the certificate signed by the  
4 CA).

5 The delegation certificate can include different components, and in one  
6 implementation the delegation certificate includes: (1) an identification of the  
7 path being delegated that is below the root of the subtree that is being managed by  
8 the directory group performing the delegation; (2) an identification of the root of  
9 the subtree delegated to the directory group performing the delegation; (3) an  
10 identification of the subtree being delegated; and (4) an identification of the  
11 members of the group to which the subtree is being delegated. The identifications  
12 of subtrees and path members can vary, and can be the actual directory names  
13 (e.g., the names of directories A, B, C, D, etc.) or alternatively identification  
14 numbers (e.g., Globally Unique Identifiers (GUIDs)). Identification numbers can  
15 be used to avoid the need to re-create delegation certificates in the event that a  
16 directory name is changed.

17 An example of delegation certificates can be seen with reference to Fig. 4.  
18 Directory group 402 obtains a certificate from a CA certifying that group 402 has  
19 authority to manage the namespace root. This certificate takes the following form:

$$\sigma_{\text{OurCA}}(\text{"Root"}, \text{GUID}_{\text{Root}}, \text{DG}_{402}) \quad (1)$$

21 where  $\sigma_{\text{OurCA}}$  indicates that the certificate has been signed by the CA "OurCA",  
22 "Root" is the name of the namespace root,  $\text{GUID}_{\text{Root}}$  is a globally unique identifier  
23 for the namespace root, and  $\text{DG}_{402}$  represents the names (or other identifiers) of  
24 the members of directory group 402.

When directory group 402 decides to delegate the subtree beginning with directory A to directory group 404, directory group 402 generates a delegation certificate to be passed to the members of directory group 404. This delegation certificate includes certificate (1) above, as well as the following certificate:

$$\sigma_{DG402}(\text{GUID}_{\text{Root}}/A, \text{GUID}_A, \text{DG}_{404}) \quad (2)$$

where  $\sigma_{DG402}$  indicates that the certificate has been signed by members of directory group 402,  $\text{GUID}_{\text{Root}}/A$  is the GUID of the subtree's root delegated to directory group 402 ( $\text{GUID}_{\text{Root}}$ ) along with the path being delegated to directory group 404 ( $/A$ ),  $\text{GUID}_A$  is a globally unique identifier of the subtree being delegated (that is, the subtree beginning with directory A), and  $\text{DG}_{404}$  represents the names (or other identifiers) of the members of directory group 404.

Similarly, when directory group 404 decides to delegate the subtree beginning with directory D to directory group 406, directory group 404 generates a delegation certificate to be passed to the members of directory group 406. This delegation certificate includes certificates (1) and (2) above, as well as the following certificate:

$$\sigma_{DG404}(\text{GUID}_A/B/C/D, \text{GUID}_D, \text{DG}_{406}) \quad (3)$$

where  $\sigma_{DG404}$  indicates that the certificate has been signed by members of directory group 404,  $\text{GUID}_A/B/C/D$  is the GUID of the subtree's root delegated to directory group 404 ( $\text{GUID}_A$ ) along with the path being delegated to directory group 406 ( $/B/C/D$ ),  $\text{GUID}_D$  is a globally unique identifier of the subtree being delegated (that is, the subtree beginning with directory D), and  $\text{DG}_{406}$  represents the names (or other identifiers) of the members of directory group 406.

In the illustrated example, delegation certificates are issued at delegation points rather than for each directory within a particular subtree. For example, a

1 delegation certificate is issued for A (the top directory in the subtree), but not for  
2 /A/B or /A/B/C.

3 Fig. 5 is a flowchart illustrating an exemplary process 500 for delegating  
4 management responsibility for a subtree to another directory group. Process 500  
5 is performed by the subtree delegation modules 254 of the computers in the  
6 directory group that are delegating management responsibility for the subtree.  
7 Initially, a group of computers to which the subtree is to be delegated is identified  
8 (act 502). A delegation certificate for the subtree is generated (act 504) and is  
9 digitally signed by one or more members of the delegating group (act 506). The  
10 digitally signed delegation certificate is then issued to the group of computers  
11 being delegated the management responsibility for the subtree (act 508).

12 Returning to Fig. 4, each computer in distributed file system 150 maintains  
13 a local cache (e.g., cache 260 of Fig. 2) mapping some subset of the pathnames in  
14 the name space to the directory group that manages that pathname. For example, a  
15 particular computer's cache may include a mapping of each of pathnames /A,  
16 /A/B, /A/B/C, /A/F, and /A/F/G to directory group 404. Different computers can  
17 have different mappings in their caches, but each typically includes at least a  
18 mapping of the namespace root to its managing directory group (directory group  
19 402).

20 Maintaining a pathname to managing directory group mapping allows a  
21 computer to perform at least some of the directory group lookup process itself  
22 locally rather than always requiring accessing the directory group managing the  
23 namespace root (and perhaps other directory groups). For example, assume that a  
24 computer desires to access a file called "foo.txt" with the pathname /A/B/foo.txt,  
25 and that the computer has in its local cache the mapping of the pathnames for

1 directory group 404. In this example, the computer can readily identify from its  
2 own local cache the members of directory group 404 that manage the files in  
3 directory B, and thus the file foo.txt. Thus, the determination of which computers  
4 to access to determine the location of the file "foo.txt" (that is, which computers  
5 manage the directory entries for pathname /A/B) is made by the computer based  
6 on the information in its cache, without having to access either directory group  
7 402 or 404 to make the determination.

8 If a computer does not have enough information in its local cache to map  
9 the entire pathname to a directory group, the computer finds the mapping for the  
10 longest prefix in the pathname that exists in its cache. The computer then accesses  
11 the directory group that manages the last directory in that longest prefix to  
12 determine the directory groups managing as much of the rest of the pathname and  
13 their delegation certificates as possible. This process of accessing directory  
14 groups and obtaining delegation certificates continues until the proper mapping is  
15 found.

16 For example, assume that a computer desires to access a file called  
17 "foo2.txt" with the pathname /A/B/C/D/foo2.txt, and that the computer has in its  
18 local cache the mapping of the pathnames for directory group 404 but not for  
19 directory group 406. The computer looks at the pathname and finds the mapping  
20 for the longest prefix in its cache that is in the pathname (/A/B/C) and accesses the  
21 directory group responsible for managing that directory, which is directory group  
22 404. The computer queries a member of directory group 404 for the delegation  
23 certificate(s) for the relevant subtrees for pathname /A/B/C/D/foo2.txt, which is  
24 the delegation certificate for directory group 406. The member of directory group  
25 404 returns this delegation certificate to the querying computer, which in turn can

1 verify the delegation certificate (e.g., based on the public key(s) of the signing  
2 computer(s)). The received delegation certificate identifies the directory group  
3 that is responsible for managing the directory /D, so the computer knows to access  
4 that directory group in order to determine where to locate the file "foo2.txt". Thus,  
5 although the determination of which computers to access to determine the location  
6 of the file "foo2.txt" involved accessing a member of directory group 404, no  
7 access to a member of directory group 402 was required to make the  
8 determination.

9 Fig. 6 is a flowchart illustrating an exemplary process 600 for looking up  
10 the directory group responsible for managing a particular pathname. Process 600  
11 is performed by directory group lookup module 222 of Fig. 2 of the computer  
12 desiring to access the pathname being looked up. Initially, a local cache of  
13 mappings to directory groups is accessed (act 602) and the mapping for the longest  
14 prefix in the pathname found in the cache (act 604). Processing then proceeds  
15 based on whether the entire pathname is mapped (act 606). If the entire pathname  
16 is mapped, then the directory group lookup process is completed (act 608).  
17 However, if the entire pathname is not mapped, then a delegation certificate(s) for  
18 the relevant subtree(s) is obtained from a member of the group managing the last  
19 mapped prefix in the pathname (act 610). The received delegation certificate(s) is  
20 then verified (act 612). If the delegation certificate does not verify correctly or  
21 cannot be obtained, then the process returns to act 610 and selects a different  
22 member of the group to query. As long as there is at least one correctly  
23 functioning member of the group managing the last mapped prefix online, the  
24 process will eventually succeed. If there is no correctly functioning member of the  
25 group managing the last mapped prefix online, then the process looks for a shorter



1 prefix that is still valid, or alternatively may return to the name space root. Once  
2 the delegation certificate chain is verified, the pathname mapping information  
3 from the certificate is added to the local cache (act 614). The process then returns  
4 to act 606, with the longest prefix now being the previously longest prefix with the  
5 new relevant subtree information concatenated thereto (for example, if the  
6 pathname is /A/B/C/D/E/F, the previous longest prefix was /A/B, and the new  
7 relevant subtree was /C/D, then the new longest prefix would be /A/B/C/D). Acts  
8 606, 610, 612, and 614 are then repeated until the entire path name is mapped. By  
9 separating the management of different directories onto different directory groups,  
10 the management responsibility is spread out over multiple different computers.  
11 This reduces the management burden on particular computers, especially those  
12 computers in the directory groups at and closest to the namespace root. For  
13 example, a particular pathname need not be parsed beginning with the root node,  
14 but rather can be picked up partway through the pathname via the local cache.

### 15 16 **Directory and File Replication and Storage**

17 Distributed file system 150 of Fig. 1 manages the storage of directory  
18 entries and the files corresponding to those entries differently. A file being stored  
19 in system 150 is replicated and saved to multiple different computers in system  
20 150. Additionally, a directory entry is generated for the file and is also saved to  
21 multiple different computers in system 150 that are part of a Byzantine-fault-  
22 tolerant group. The directory entry is saved to more computers than the file is  
23 saved to, as discussed in additional detail below.

24 The different treatment for storage of files and directory entries described  
25 herein can be used in conjunction with the hierarchical storage structure discussed

1 above. However, the different treatment for storage of files and directory entries  
2 described herein can also be used in systems that do not employ a hierarchical  
3 storage structure.

4 A Byzantine-fault-tolerant group is a group of computers that can be used  
5 to store information and/or perform other actions even though a certain number of  
6 those computers are faulty (compromised or otherwise unavailable). A computer  
7 can be compromised in a variety of different manners, such as a malicious user  
8 operating the computer, a malicious program running on the computer, etc. Any  
9 type of behavior can be observed from a compromised computer, such as refusing  
10 to respond to requests, intentionally responding to requests with incorrect or  
11 garbage information, etc. The Byzantine-fault-tolerant group is able to accurately  
12 store information and/or perform other actions despite the presence of such  
13 compromised computers. Byzantine groups are well-known to those skilled in the  
14 art, and thus will not be discussed further except as they pertain to the present  
15 invention.

16 It is known to those skilled in the art that for certain types of computations  
17 in order to be able to operate correctly despite a number of failed computers  $f$  (a  
18 failed computer may be compromised or otherwise unavailable, such as powered  
19 down), the Byzantine-fault-tolerant group should include at least  $3f+1$  computers.  
20 In distributed file system 150, the directory entries are stored on the  $3f+1$   
21 computers of a Byzantine-fault-tolerant group, while the file itself is stored on  $f+1$   
22 computers (which may be one or more of the same computers on which the  
23 directory entry is stored).

24 Fig. 7 illustrates the exemplary storage of a file and corresponding directory  
25 entry in a serverless distributed file system. File system 700 (e.g., a serverless

distributed file system 150 of Fig. 1) includes twelve computers 702, 704, 706, 708, 710, 712, 714, 716, 718, 720, 722, and 724. Assuming that the designer of system 700 desires to be able to tolerate two computer failures, the Byzantine-fault-tolerant group should include at least seven  $((3 \cdot 2) + 1)$  computers. Byzantine group 726 is illustrated including computers 702 – 714.

When a file 728 is to be stored in file system 700, a corresponding directory entry 730 is stored by the computers in the appropriate directory group (the directory group responsible for managing the directory the file is stored in, based on the pathname of file 728). The directory group in Fig. 7 for directory entry 730 is Byzantine group 726, so the directory entry 730 is stored on each correctly functioning computer 702 – 714 in Byzantine group 726. Thus, directory entry 730 is stored on up to seven different computers. File 728, on the other hand, is replicated and stored on each of three computers (computers 716, 720, and 724). As illustrated, the computers on which file 728 are stored need not be, and typically are not, in Byzantine group 726 (although optionally one or more of the computers on which file 728 are stored could be in Byzantine group 726).

Each directory entry includes the name of the corresponding file, an identification of the computers that the file is stored at, and file verification data that allows the contents of the file to be verified as corresponding to the directory entry. The file verification data can take a variety of different forms, and in one implementation is a hash value generated by applying a cryptographically secure hash function to the file, such as MD5 (Message Digest 5), SHA-1 (Secure Hash Algorithm-1), etc. When a file is retrieved from storage, the retrieving computer can re-generate the hash value and compare it to the hash value in the directory entry to verify that the computer received the correct file. In another

1 implementation, the file verification data is a combination of: a file identification  
2 number (e.g., a unique identifier of the file), a file version number, and the name  
3 of the user whose signature is on the file.

4 Fig. 8 is a flowchart illustrating an exemplary process for storing a file in a  
5 serverless distributed file system. Initially, a new file storage request is received at  
6 a client computing device (act 802). The client encrypts the file and the file name  
7 and generates the file contents hash (act 804). The client sends the encrypted file  
8 name and file contents hash to the appropriate Byzantine-fault-tolerant directory  
9 group along with a request to create a directory entry (act 806). The directory  
10 group validates the request (act 808), such as by verifying that the file name does  
11 not conflict with an existing name and that the client has permission to do what it  
12 is requesting to do. If the request is not validated then the request fails (act 810).  
13 However, if the request is validated, then the directory group generates a directory  
14 entry for the new file (act 812). The directory group also determines the replica  
15 set for the new file and adds the replica set to the newly generated directory entry  
16 (act 814). Replicas of the file are also generated (act 816), and saved to multiple  
17 computers in the file system (act 818).

18 By storing the directory entries in a Byzantine group, and including file  
19 verification data in the entries, fault tolerance is maintained (up to  $f$  failures).  
20 However, storage space requirements and Byzantine operations are reduced by  
21 storing files separately from directories and not using Byzantine operations to  
22 access them. For example, directory entries may be on the order of one hundred  
23 bytes, whereas the file itself may be on the order of thousands or even billions of  
24 bytes.

## Directory and File Lock Mechanism

Each object (e.g., directory and file) in distributed file system 150 of Fig. 1 has associated with it a set of leased locks. These locks are used to determine, based on the type of operation an application desires to perform, whether the application can open a directory or file to perform that operation. A lock can be viewed as a lease with a particular time span that depends on the type of lock and the level of contention. For example, the time span on a write lock may be a few minutes, while the time span on a read lock may be as long as a few days. When an application desires to perform an operation(s) on an object, the client computer on which the application is executing looks to see if it already has the necessary locks to perform the operation(s). If not, it requests the appropriate lock(s) from the directory group responsible for managing that object. Once the application has finished performing the desired operation, it can optionally release the lock(s) it acquired or keep it until it automatically expires or is recalled by the managing directory group.

For a particular directory, the Byzantine-fault-tolerant group that implements the directory controls the locks for: all files in the directory; the names of any subdirectories of the directory; and the right to delete the directory itself. The lock mechanism attempts to grant broad (coarse granularity) locks on appropriate files and directories to a requesting client computer so that the client computer can process many reads and/or updates with a single Byzantine lock acquisition rather than requiring multiple Byzantine messages for lock acquisitions.

In the illustrated example, the lock mechanism employs ten different locks: Read, Write, Open Read, Open Write, Open Delete, Not Shared Read, Not Shared

1 Write, Not Shared Delete, Insert, and Exclusive. The Read and Write locks are  
2 used to control access to the data in the objects (e.g., the contents of a file). The  
3 Open Read, Open Write, Open Delete, Not Shared Read, Not Shared Write, and  
4 Not Shared Delete locks are used to control the opening of the objects. The Insert  
5 and Exclusive locks are special-use locks. These ten locks are discussed in more  
6 detail below. Depending on the operation an application desires to perform, the  
7 appropriate ones of these locks are requested by the application.

8 Read Lock. The Read lock is requested by an application so that the  
9 application can read the associated file. The Read lock, in conjunction with the  
10 Write lock, allows the directory group to keep data in the file consistent.

11 Write Lock. The Write lock is requested by an application so that the  
12 application can write to (also referred to as update) the associated file. The Write  
13 lock, in conjunction with the Read lock, allows the directory group to keep data in  
14 the file consistent.

15 When an application desires to open an object, the directory group performs  
16 two checks: (1) are the modes the application is asking for going to conflict with  
17 another application that has already opened the object; and (2) are the operations  
18 that the application is willing to share the object for going to conflict with what  
19 another application has already opened the object for and indicated it is willing to  
20 share the object for. Six of the ten locks are directed to supporting this checking:  
21 Open Read, Open Write, Open Delete, Open Not Shared Read, Open Not Shared  
22 Write, and Open Not Shared Delete. These locks are used to grant an application  
23 the ability to open an object, but do not necessarily guarantee that the data for the  
24 object can be obtained (the Read lock or Write lock (depending on the type of  
25 operation the application desires to perform) is obtained to access the data).

1        Open Read Lock. The Open Read lock is requested by an application to  
2 allow the application to open the associated object for reading.

3        Open Write Lock. The Open Write lock is requested by an application to  
4 allow the application to open the associated object for writing.

5        Open Delete Lock. The Open Delete lock is requested by an application to  
6 allow the application to open the associated object for deleting.

7        Open Not Shared Read Lock. The Open Not Shared Read lock is requested  
8 by an application when the application is not willing to share the ability to read the  
9 object with any other application.

10       Open Not Shared Write Lock. The Open Not Shared Write lock is  
11 requested by an application when the application is not willing to share the ability  
12 to write to the object with any other application.

13       Open Not Shared Delete Lock. The Open Not Shared Delete lock is  
14 requested by an application when the application is not willing to share the ability  
15 to delete the object with any other application.

16       The other two locks that are supported are the Insert Lock and the  
17 Exclusive Lock.

18       Insert Lock. The Insert lock is requested by an application to create a  
19 particular name for an object in a directory. Granting of the Insert lock gives the  
20 application permission to create the object with the particular name. The Insert  
21 lock conflicts with another Insert lock with the same object name, and with an  
22 Exclusive lock on the directory.

23       Exclusive Lock. The Exclusive lock is requested by an application to  
24 obtain all of the previously discussed nine locks, including an Insert lock on each  
25 possible name that could exist (but does not already exist) in the directory. An

Exclusive lock on a directory does not imply Exclusive locks on the files or subdirectories in the directory, but rather only on the directory's namespace. The Exclusive lock conflicts with each of the previously discussed nine locks.

Various conflicts exist between the various different locks. Table I is a conflict matrix illustrating the conflicts between locks in one exemplary implementation. The following abbreviations are used in Table I: Ins (Insert), Excl (Exclusive), O-R (Open Read), O-W (Open Write), O-D (Open Delete), O-!R (Open Not Shared Read), O-!W (Open Not Shared Write), and O-!D (Open Not Shared Delete). An "X" in a cell of Table I indicates a conflict between the corresponding two locks – for example, Open Read conflicts with Open Not Shared Read but does not conflict with Open Not Shared Write.

Table I

	Ins	Read	Write	Excl	O-R	O-W	O-D	O-!R	O-!W	O-!D
Ins	X	X	X	X						
Read	X		X	X						
Write	X	X	X	X						
Excl	X	X	X	X	X	X	X	X	X	X
O-R				X				X		
O-W				X					X	
O-D				X						X
O-!R				X	X					
O-!W				X		X				
O-!D				X			X			



Fig. 9 is a flowchart illustrating an exemplary process for determining whether to allow a particular object to be opened. The process of Fig. 9 is implemented by the directory group responsible for managing the particular object. In the process of Fig. 9, it is assumed that the client requesting to open the particular object does not already have the necessary lock(s) to open the object as desired. Initially, a request to access an object with particular locks identified is received (act 902). A check is made by the directory group as to whether the modes implied by the selected locks conflict with locks that have been granted to a different client (act 904). For example, if the request is a request to open an object for reading, but another application has already opened the object with the Not Shared Read lock, then the mode (open read) implied by the selected lock conflicts with another application that has already opened the object. Because the directory group knows only if it has issued a conflicting lock to a client, but not whether the client is currently using the lock to allow an application access to an object, in some cases making the check in act 904 requires asking a client that currently holds a lock is willing to give it up.

If the check in act 904 identifies no conflict, then the requested locks are granted to allow the application to open the file with the selected locks (act 906), so the request in act 902 is granted. The fact that these locks have been granted, and the clients to which they have been granted are then saved by the directory group (act 908) so that they can be used to determine conflicts for subsequent requests, and can be used to attempt recalls of locks when necessary.

However, if the check in act 904 identifies a conflict, then a request(s) is issued to the client(s) holding the conflicting locks to return them (act 910). A

check is then made as to whether all of the requested locks were returned (act 912). If all of the requested locks were returned, then the requested locks are granted to allow the application to open the file with the selected locks (act 906), and the locks recorded (act 908). On the other hand, if all of the requested locks were not returned, then the open request is denied by the directory group (act 914).

In an attempt to improve performance when only one client computer accesses some region of the namespace, the file system 150 may issue a lock with broader scope than an application executing on the client requests, under the assumption that the application (or client) is likely to request additional related locks in the near future. For example, if an application opens file /A/B/C/foo.txt, the client requests a lock for this file. If the directory group grants the lock, it may upgrade the lock to a directory lock on /A/B/C (e.g., if, based on past performance, the directory group determines that conflicts on the directory are rare). If the application then opens another file in the same directory, the client can open the file without needing to request another lock from the directory group.

If a client's lock request conflicts with an existing lock granted to another client, the directory group may attempt to downgrade the earlier-issued lock to one that will not conflict with the new request at act 910 (e.g., rather than denying the request in act 914). Since lock upgrades result in clients holding locks that they did not request, lock downgrades typically have a non-trivial likelihood of success. If the lock recall fails, then the request is denied.

Various operations can be performed on objects in a file system. Table II below describes several of the more common operations and what locks are requested by an application in order to perform the operations.

Table II

Operation	Description
Read Object	A request to read a directory or file. Requires an Open Read lock for the object followed by a Read lock. Optionally, if desired, the application may request any of the Open Not Shared locks.
Write/Update Object	A request to write to a file. Requires an Open Write lock for the object followed by a Write lock. Optionally, if desired, the application may request any of the Open Not Shared locks.
Delete File	A request to delete a file in a directory. Requires the Open Delete and Write locks. Usually the application will also request all of the Open Not Shared locks.
Delete Directory	A request to delete a directory. Requires an Exclusive lock for the directory. Directories may only be deleted when they are empty.
Rename Directory	A request to rename a directory. Requires an Exclusive lock on the parent directory (the directory for which the directory being renamed is a subdirectory), and an Insert lock for the new directory name in the destination directory. If the rename is across directories then the insert lock will be required for the new parent directory.
Rename File	A request to rename a file in a directory. Requires a Write lock on the file, and an Insert lock for the new name in the directory (which may be a different directory, if the rename is across directories).
Create Object	A request to create a new file or directory. Requires an Insert lock for the new name.

Any changes made to a file are made locally by the computer and then the file (after being encrypted) is pushed back to the directory group responsible for managing the file. This information is stored to the various computers in the directory group, and the updated file is stored to the appropriate computers.

## Conclusion

Although the description above uses language that is specific to structural features and/or methodological acts, it is to be understood that the invention

1 defined in the appended claims is not limited to the specific features or acts  
2 described. Rather, the specific features and acts are disclosed as exemplary forms  
3 of implementing the invention.

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25